Test Driven Development

with a MATLAB example

Mike.Park@NASA.Gov

Computational AeroSciences Branch

FUN3D Software Development Team

2 April 2009

Sidebar before we start. . . Software Versioning System

- Git - `http://git-scm.com/`
- Subversion - `http://subversion.tigris.org/`
- CVS - `https://www.cvshome.org/`
- Often overlooked or under emphasized
- Zeroth principle of software engineering
- Learning to work with it and not against it is key
  - personal and team programming (glue)
- Safety net
- Large impact on "Truck Number"
- Convenient for accounts on multiple machines
- Required for automated testing
- Not just for software anymore
  - presentations, configuration files, home accounts

Why? (a CFD Developer's Perspective)

Multidisciplinary problems require multiple discipline experts, a large infrastructure, and standard interfaces

Reduce time from concept to application for vehicles and algorithms

Mobility to respond to unforeseen challenges and increase software lifespan

Research capabilities in a "production" code

Infrastructure to evaluate algorithms on large problems

Flexibility for implementing research algorithms

Stability to suit time-sensitive application needs and to release to outside customers

Avoid being encumbered by high-ceremony software development process

Software Testing

All of these can (and should) be automated!

Programmer's I want a function that adds vectors, does $f([1,2],[3,4])$ return $[4,6]$? (unit tests)

Integration Does my whole system compile and work together?

Regression My code gave result x yesterday, does it give answer x today?

Verification My code is supposed to exactly give this analytic result, does it?

Validation Does my code give the same answer as a bench top test or fight test measurement?

Unit Testing and Test Driven Development

- Seems trivial at first
- Hard to imagine benefit until the first major refactoring or code simplification is experienced
- Gains power as the number of tests and their coverage increases
- Your own custom debugger
- Provides a clear completion to an implementation task
- Code with a failing test is much easier to fix or extend
- Inventing the tests required is generally harder
- Code that is easy to test is often simpler and easier to read, understand, and extend
- Creating tests brings the design to the forefront; design is difficult, but it is easiest in small increments

Test Driven Development Rhythm

Add a broken test

that covers the capability that you want to add or bug you wish to fix

Run all tests, make sure that new one fails

to test the test and framework

Write simplest code possible

to satisfy the test, don't get a head of yourself

Run all tests

to make sure that the new code fixes the new test and does not break existing capability

Refactor

to clean up new code and remove duplication

Unit Testing Frameworks

- Goals
  - Interface must allow for the easy creation and management of tests
  - Minimal additional effort over writing the actual code (benefit–cost)
  - Enable programmers to experience the benefits of test-first programming as soon as possible
  - Legible as documentation
- Flavors
  - `http://c2.com/cgi/wiki?TestingFramework`
  - Full featured (scripting languages)
  - Minimalistic (four lines of code)
  - Wrap code to utilize scripting language framework
  - “Roll your own”

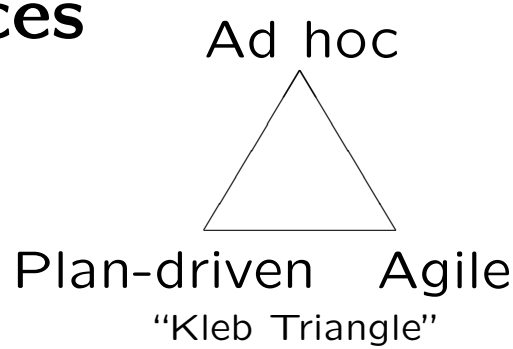
Pair Programming

- Two programmers work together at one computer on the same task
  - One person driving the other navigating
  - Switch jobs when ever person driving can not follow the navigator (navigator feels frustrated or driver feels lost)
- Pairing station really helps
- Great for debugging, knowledge transfer
- Impeded by scheduling conflicts (i.e., the other stuff at NASA)

MATLAB Example with Pairing

Appendix on Software Development Practices

Software development practices



"Kleb Triangle"

- Ad hoc
  - "Code and Fix"
- Plan-driven
  - Predictive, "Big up front design"
  - Delivering to the original contract
  - Capability Maturity Model (CMM), CMMI
- Agile
  - Adaptive, "Evolutionary design"
  - Recognizes software development an empirical process that can not always be defined
  - Extreme Programming

The Agile Manifesto values

individuals and interactions

over processes and tools

working software

over comprehensive documentation

responding to change

over following a plan

customer collaboration

over contract negotiation

Extreme Programming values

- communication
- simplicity
- feedback
- courage

Extreme Programming Practices

Sustainable pace productivity does not increase with hours worked.

Metaphor guide all development with a simple shared story of how the whole system works.

Coding standard write all code in accordance with rules emphasizing communication through the code.

Collective ownership anyone can change any code anywhere in the system at any time.

Continuous integration integrate and build the system many times a day.

Small releases release new versions on a very short cycle.

Extreme Programming Practices (concluded)

Test-driven development any program feature without an automated test simply does not exist.

Refactoring restructure the system without changing its behavior.

Simple design system should be designed as simply as possible at any given moment.

Pair programming two programmers work together at one computer on the same task.

On-site customer include a real, live user on the team.

Planning game combine business priorities and technical estimates to determine scope of next release.

FUN3D Development

Sustainable pace work ∼40 hour weeks.

Metaphor engineering and scientific vocabulary (ρ, u, v, w).

Coding standard published to aid portability, automated parsing, and collective ownership.

Collective ownership routinely fix minor bugs or extend methods created by other people. Anyone is allowed to modify any file at anytime through SVN.

Continuous integration very slow: Linux builds every 2-3 hours, SGI builds every 8-9 hours.

Small releases application members of the team use (SVN), formally 2-3 times a year

FUN3D Development (concluded)

Test-driven development limited use in flow solver, extensive use in scripting and grid adaptation.

Refactoring done only when necessary, extremely difficult, painful, and nerve-racking without unit tests.

Simple design born as a result of refactoring and pair programming.

Pair programming limited to mostly debugging, knowledge transfer; impeded by scheduling conflicts.

On-site customer research: we are our own customers?

Planning game comes more naturally in pair programming scheduling.

Communication

Collocation

Email list

WikiWikiWeb – `http://c2.com/cgi/wiki?WelcomeVisitors`

Scrum status meetings

What they **did** since last meeting

What they will **do** by next meeting

What got **in the way** (impediments)

Quick and efficient meeting style

Reduces the worst management sin (wasting people's time)

The impediments is the often the hardest to express, but the most important.